46th AIAA Aerospace Sciences Meeting and Exhibit
7 - 10 January 2008, Reno, Nevada

AIAA 2008-481

46th AIAA Aerospace Sciences Meeting and Exhibit, 7-10 January 2008

A Methodology for CFD Acceleration through Reconfigurable Hardware

Esther Andrés*
Instituto Nacional de Técnica Aeroespacial, Torrejón de Ardoz, Madrid, 28850, Spain

Carlos Carreras† and Gabriel Caffarena‡
Universidad Politécnica de Madrid, Madrid, 28040, Spain

Maria del Carmen Molina§
Universidad Complutense de Madrid, Madrid, 28040, Spain

Octavio Nieto-Taladriz¶
Universidad Politécnica de Madrid, Madrid, 28040, Spain

Francisco Palacios‖
Instituto Nacional de Técnica Aeroespacial, Torrejón de Ardoz, Madrid, 28850, Spain

The long computation times required to simulate complete aircraft configurations remain as the main bottleneck in the design flow of new structures for the aeronautics industry. In this paper, the novel application of specific hardware (FPGAs) in conjunction with conventional processors to accelerate CFD is explored in detail. First, some general facts about application-specific hardware are presented, placing the focus on the feasibility of the development of hardware modules (FPGAs based) for the acceleration of most time-consuming algorithms in aeronautics analysis. Then, a practical methodology for developing an FPGA-based computing solution for the quasi 1D Euler equations is applied to the Sod's 'Shock Tube' problem. Results comparing CPU-based and FPGA-based solutions are presented, showing that speedups around two orders of magnitude can be expected from the FPGA-based implementation. Finally, some conclusions about this novel approach are drawn.

Nomenclature

CFD	Computational Fluid Dynamics
HPC	High-performance Computing
RCC	Reconfigurable Computing
IP	Intellectual Property
FPGA	Field Programmable Gate Array
CPU	Central Processing Unit
GPU	Graphical Processor Unit
ASIC	Application Specific Integrated Circuit
GPGPU	General Purpose Programming on GPU

*Research Scientist, Departamento de Aerodinámica y Propulsión.
†Associate Professor, Departamento de Ingeniería Electrónica, E.T.S.I. Telecomunicación.
‡Assistant Professor, Departamento de Ingeniería Electrónica, E.T.S.I. Telecomunicación.
§Assistant Professor, Departamento de Arquitectura de Computadores y Automática, Facultad de Informática.
¶Full Professor, Departamento de Ingeniería Electrónica, E.T.S.I. Telecomunicación.
‖Research Scientist, Departamento de Aerodinámica y Propulsión, AIAA Member.

RTL Register Transfer Level
VHDL Very High Speed Integration Hardware Description Language

I. Introduction and motivation

Scientific Computing with its core ingredients, modeling, simulation and optimization, is regarded by many as the third pillar of science, complementary to experiment and theory. In aeronautics engineering, the consistent use of mathematical and computational methods to simulate complex processes has become indispensable to save energy, reduce costs and pollution, and to increase safety. However, the high complexity of some of these processes frequently implies very long computation times. In particular, the analysis of a complete aircraft configuration including all relevant payload elements and flight components, even using a Reynolds-Averaged Navier-Stokes (RANS) modeling, at present still requires a huge computational effort. Thus, reducing the time for aerodynamic analysis is one of the most important challenges of current research in CFD.

In computer science, Moore's law predicts that the speed achievable on a single chip doubles every 18 months, and this statement has held true for decades (even nowadays). But unfortunately, in a near future, the ever-increasing transistor density will no longer deliver comparable improvements in performance. The HPC community had detected the situation and new research lines in computer science are being intensively explored. They include multicore systems, specialized processors and hardware, and heterogeneous computing architectures, in which conventional and specialized processors work cooperatively. In particular, this paper focuses on the application of FPGA-based hardware modules to improve the performance of CFD algorithms currently executing on general purpose platforms.

A typical simulation platform in the aeronautics industry consists of a CFD specific software application (written in a high level language), and a general purpose hardware (clusters of processors or shared-memory supercomputers), on which the application is executed. The kernel of most current computer architectures is a general purpose CPU. The CPU consists on a large number of logic gates that work as on-off switches and are permanently wired in a variety of fixed circuits that implement, in a binary logic framework, the whole gamut of functions needed for its general purpose operation. At any particular time, only a small fraction of the total number of gates may be gainfully employed, while the idle gates consume power (the system clock usually runs at a very high frequency), generate heat and do not do useful work. Therefore, the main disadvantages of this generic approach are poor resource utilization, high power consumption and, ultimately, no room for performance improvements based on application-specific features.

In CFD applications, the increasing demands for accuracy and simulation capabilities produce an exponential growth of the required computational resources. This situation calls for new simulation platforms based on heterogeneous architectures in which conventional processors and specific hardware modules work together, instead of the classical approach based on general purpose hardware.

The aeronautic industry requirements in the near future will not be accomplished by the usual evolution of current processors and architectures. Indeed, it is estimated that to obtain "engineering-accuracy" predictions of surface pressures, heat transfer rates, and overall forces on asymmetric and unsteady turbulence configurations, the grid size will have to be increased to a minimum of 50-80 million nodes. This means that the simulation of a complete aircraft configuration will require several days to obtain solutions in a high performance cluster with hundreds of processors, so an exponential increment of the computational requirements is estimated.[1] Therefore, it will be necessary to introduce new concepts in typical simulation platforms in order to satisfy these demands and to face more complex challenges.

I.A. Accelerating scientific applications through specific hardware

In this section, we introduce three of alternatives for the acceleration of scientific applications using specific hardware devices. From the most general-purpose to the most specific-purpose technology, the alternatives to be considered are:

1. **Graphics Processing Unit (GPU).** In modern graphics cards, GPUs are now being considered as a cost effective computation workhorse. Multiple independent pipelines and vector arithmetic make GPUs a powerful parallel processing platform. Recent additions of full programmability and IEEE-standard floating point arithmetic allow graphics cards to perform general purpose computations that

have traditionally not been executable in GPUs, due to their wordlenght requirements. In the current state of the art, GPUs can perform tens of billions of floating point calculations per second. And because all this power is available at a very low cost, the scientific community is currently devoting a significant amount of effort to develop new algorithms to carry out on GPUs.

GPUs are well suited for CFD applications since CFD requires intensive vector-based mathematics, specially for tasks such as mesh deformation or sparse matrix solving,[2] linear algebra operations,[3] and fast Fourier transforms.[4] However, there are limitations when more specific algorithms are to be implemented due to the generic nature of GPUs (number of registers, memory...), and the real performance obtained is often lower than the peak performance expected.

2. **Field Programmable Gate Array (FPGA).** Reconfigurable computing is intended to bridge the gap between hardware and software. Dynamically reconfigurable supercomputers can potentially contribute to important value metrics, including time-to-solution, by reducing design cycle time and porting costs, reducing electrical power usage and enhancing fault-tolerance, for example. The flexibility of programmable gate arrays (FPGA) raises the possibility of a meta-architecture; morphing hardware configurations with software as needed to improve efficiency, robustness, security and capability on-the-fly.[5–7]

   FPGAs have evolved rapidly in the last decade. Embedded (or "soft") Intellectual Property (IP) cores representing common, low-complexity microprocessors and microcontrollers are available from third-party suppliers. Using such cores in combination with other related peripheral cores makes it possible to create complete systems.

   The two major advantages of FPGAs over general-purpose sequential processors are that they provide support to exploit the implicit parallelism of each algorithm to its full extent and that they are reconfigurable, thus they can be adapted to the specific needs of each individual algorithm. Therefore, despite the fact that the clock frequency of general-purpose processors is about 20 times higher than of the typical FPGA implementations, significant gains can be expected from the use of reconfigurable systems.

3. **Application-Specific Integrated Circuit (ASIC).** An alternative to general purpose hardware are ASICs, that are integrated circuits made for only one specific function. The advantages of ASICs are high performance, less silicon area, and low power consumption. The disadvantages of an ASICs based approach are that it is not flexible (due to its fixed connections and functionalities) and it is much more expensive in design and fabrication. In addition, small changes in the designed circuit (for a new functionality or optimization) imply a new complete fabrication process. This huge cost makes ASICs not practical for some problems.

I.B. Specific Purpose Hardware (FPGAs), its current industrial application

Applications of FPGAs in industry cover a broad range of areas including Digital Signal Processing (DSP), aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics,[8] computer hardware emulation and a growing range of other areas[9–11] . FPGAs especially find application where algorithms can make use of the massive parallelism offered by their architecture.

In the biomedical sector, for example, FPGAs have been broadly used for real time image processing[12] . Three-dimensional ultrasonic imaging, especially the emerging real-time version of it, is particularly valuable in medical applications such as echocardiography, obstetrics and surgical navigation. A known problem with ultrasound images is their high level of speckle noise and anisotropic diffusion. However, due to its arithmetic complexity and the sheer size of 3D ultrasound images, it is not possible to perform on line, real-time anisotropic diffusion filtering using standard software implementations. This task can be done with an FPGA-based architecture that allows performing anisotropic diffusion filtering of 3D images at acquisition rates coupled with a multigrid algorithm. This enables the use of filtering techniques in real-time applications, such as visualization, registration and volume rendering[13] . Therefore, in this sector, the application of FPGAs permits to deal with data at the rates needed.

With respect to the acceleration of critical functions, various special-purpose application hardware accelerators have been built. The GRAPE family of machines[14] is perhaps the most noteworthy, having won three

Gordon Bell prizes for solving astrophysics calculations. The recent GRAPE-6 machine has 64 processors plus 2,048 ASICs and has a computational power of 64 TFLOPS. Each ASIC clocks at 88 MHz and has 6 pipelines with 57 floating-point operations per pipeline yielding 30.1 GFLOPS per device.

II. Feasibility of FPGAs application in CFD

Practical aeronautical design problems have to be executed on supercomputers or clusters which require days to obtain a complete polar computation with well convergence solutions. Therefore, the aerospace industry has the challenge of reaching powerful computational resources to simulate and optimize complex configurations in a reasonable period of time.

Hardware acceleration gives best results, in terms of overall acceleration and value for money when applied to problems in which:

- Condition A. A great amount of the computational effort is concentrated in a small portion of the whole code, and the computations have to be performed several times for a huge set of data.
- Condition B. The communication and I/O times are small with respect to the total computational cost.

The aeronautical analysis HW-SW application presented here focuses on implementing the most time consuming algorithms involved directly into hardware by using reconfigurable devices (FPGA). The selection of FPGAs, allows the user to build a temporary custom circuit for solving a CFD problem (or a part of the CFD chain). Since the size, number, and type of function units (e.g. add, sub, mult, div) are defined by the programmer, algorithms can be parallelized at the instruction level, so programmers can have many of these instructions executed in the same clock cycle.

Other factors that also contribute to such gains are: the streaming of data from memory, the overlap of control and data flow, and the elimination of some instructions on the FPGA. In this way, users can achieve high performance gains for certain types of problems for which traditional parallel processing techniques have offered few advantages.

When considering aerodynamics simulations, the numerical solution of the flow equations is based on a flux interchange between the discrete volumes that represent the physical domain. The numerical flux computation for each discrete volume requires several hundred floating-point operations, and each operation is repeated several million times in a real aeronautical simulation (Condition A). Also, this computation can be pipelined to further exploit fine-grained parallelism. This so called fine-grained parallelism comes from the fact that some specific hardware modules are synthesized to perform several operations simultaneously.

On the other hand, the numerical solution of the flow has data locality. This means that the computation to be performed at a certain location of the domain depends only on data that is located in a small neighborhood around it. This is particularly interesting for parallel computing, where the whole computational domain is decomposed into a certain number of subdomains, and each subdomain is assigned to a different processor. Communication between the processors is required because the solutions on the subdomains depend on each other, but only on the subdomain boundaries. This is the reason why communication costs are small with respect to the overall cost (Condition B).

To sum up, there are a number of characteristics of FPGA-based systems that make them well suited for accelerating critical functions in aeronautical analysis configurations.

1. The control flow and data flow graphs for the inner-loop calculations can be readily mapped to pipelined hardware implementations. By coupling the FPGA device with high-bandwidth external memory, it is possible to achieve a performance of several GFLOPS sustained on an FPGA versus the magnitude of only hundreds of MFLOPS sustained on a conventional processor.
2. Numerical methods applied to solve governing flow equations only include multiplications, additions, divisions and other simple operations, even in complex cases.
3. The array data cache may be designed and optimized to match the specific characteristics of the CFD algorithms. The ability to tune the implementation of the array data cache to match the specific function's characteristics allows to keep the inner-loop calculation pipelines busy.

4. The total memory bandwidth and the amount of memory available can be configured in the specific system to match the needs of the algorithms as well as the desired 3D mesh size. If an optimal implementation is required, it is possible to build a board with a custom memory size and other specific features.

5. The code used in the inner-loop calculations is relatively stable and the investment required to build an RCC-based implementation can be leverage across a large portion of the CFD application code.

As a conclusion, aeronautical optimal design problems gather all the desirable characteristics to be executed on a platform with a heterogeneous architecture (specific software - specific hardware accelerator - generic hardware platform).

II.A. New level of parallelism on CFD applications with PC clusters

Current simulation platforms use the so called coarse-grained parallelism. In this kind of parallelism the computational domain is divided in several subdomains that are executed in different processors (for example in a PC cluster architecture). Therefore, the parallelism here is basically obtained from the domain decomposition and the execution of the same program for different set of data at the same time. On the other hand, this coarse-grained parallelism could be improved with a new level of fine-grained parallelism by accelerating the set of operations assigned to each local processor (parallelism in the hardware functional units that provide several results at the same time).

The first objective related to hardware acceleration is the development of a prototype of a hardware-software simulation platform for studying the feasibility of this mixed approach. The goal is to build an FPGA-powered cluster, where one FPGA board is included in each computational node (see figure 1). Therefore, such system will support coarse grain-parallelism among PCs in the cluster and fine-grain parallelism at the level of the FPGA board in each PC.

All computational nodes in the cluster are FPGA-powered, so load balance has to be taken into account. This means that the subdomain decomposition process has to assign to the FPGA-powered computational nodes an adequate workload in order to support efficient synchronization in the overall system. Due to the heterogeneous nature of the architectures existing in a cluster platform, including differences in performance between the general-purpose CPUs, special considerations about global synchronization have to be explored.

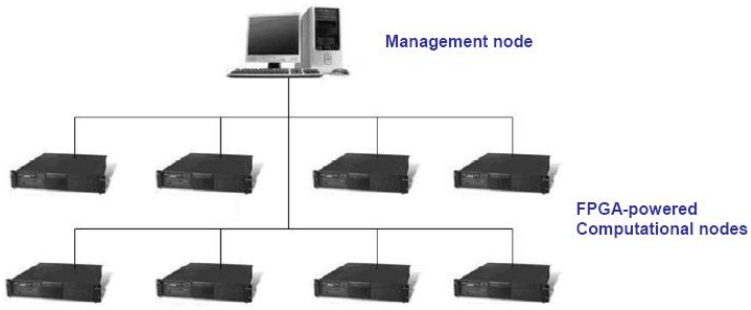

Figure 1. FPGA powered cluster.

II.B. Requirements to achieve high speedups

The first requirement in assessing the value of FPGA acceleration for an application is to understand how the FPGA is to be connected to the computational system.

Currently, engineers are placing groups of FPGAs on dedicated circuit boards which, in turn, are attached to standard microprocessor-based systems. This allows engineers to use the flexibility of the sequential microprocessor for more conventional tasks, while offloading work to the attached board. The FPGA board may filter data coming into the system and then send the results to the microprocessor for further analysis.

The current crop of FPGAs from chip vendors offers enough logic to support dozens of 64-bit function units. For example, by dividing the logic requirements for the 64-bit floating point cores from Xilinx[15] into the available logic on the Virtex-4 LX200,[16] it can be observed that, theoretically, this chip could support about 70 floating point multiplication units or 128 floating point addition units.[17]

However, engineers are still saddled with two key issues:

1. The communication rate between PC-host and FPGA, that is the bandwidth, because data are located in the microprocessor's memory and need to be moved across an interface to the FPGA. To address

this problem, vendors have placed FPGAs on much faster PCI-X, PCI-e and HyperTransport (HT) interfaces in contrast with early FPGA architectures that typically were placed on PCI interfaces. PCI-X runs at 133 MHz and is 64-bits wide, yielding a bandwidth of 1064 MB/s. In the PCI-e 1.1 specification, each lane carries 250 MB/s in each direction. Alternatively, HT is an open protocol for attaching devices directly to the processor. It can support bandwidths of up to 3.2 GB/s in each direction. Using HT, the FPGA can be attached directly to the processor with minimal latency and optimal bandwidth.

2. The limited area available in current chips that limits the complexity of the designs to be synthesized. Engineers also employ alternative techniques to increase the advantages of the FPGA, such as trying reduced precision based on fixed-point arithmetic to decrease the amount of area required and data transferred, and to squeeze more operations per clock cycle out of the FPGA. Using single precision, 32-bits, halves the amount of area requirements and allows more than twice as many function units and words in BRAM on the FPGA. Fixed-point arithmetic not only provides improved performance and reduced area, but it also has the advantage of reduced power consumption. There are some techniques to convert a floating point algorithm into a fixed point algorithm with the same accuracy[18–20] .

III. FPGA application: Sod's "Shock Tube" problem

This section is devoted to introduce a specific time-consuming algorithm that can be accelerated using the proposed approach to obtain an important improvement in the performance. First, a brief introduction to the selected mathematical model to be solved is presented and the numeric algorithm is described in detail. Then, the design of the accelerator module is specified step by step, and finally, some tests are performed and discussed.

III.A. Step 1: Model Description

III.A.1. Step 1.1: Sod's 'Shock Tube' problem

The 1D Euler equations for gas dynamics are considered as example of application

$$\partial_t U + \partial_x F = 0, \quad \text{on } 0 \leq x \leq 1.0 \tag{1}$$

with initial conditions from Sod's shock tube problem

$$\begin{cases} (\rho, \rho v, \rho e) = (1.0, 0, 2.5) & \text{if } x \leq 0.5, \\ (\rho, \rho v, \rho e) = (0.125, 0, 0.25) & \text{if } x > 0.5, \end{cases} \tag{2}$$

where the conservative variables are $U = (\rho, \rho u, \rho e)^T$, the convective flux is $F = (\rho u, \rho u^2 + p, \rho u h)^T$, ρ represents the gas density, v represents the gas velocity, e represents the total energy of the gas and h is the enthalpy. The system of equations is complemented by a state equation for an ideal gas which links the variables

$$h = e + \frac{p}{\rho} = \frac{\gamma}{\gamma - 1}\frac{p}{\rho} + \frac{1}{2}v^2 \tag{3}$$

where γ stands for the adiabatic gas index. For an ideal gas $\gamma = 1.4$.

The solution starts with a hot high density gas in the region $0 \leq x \leq 0.5$, and a cold low density gas in the region $0.5 < x \leq 1.0$. The gas is initially at rest. At $t = 0$ the diaphragm separating the two regions is removed, causing a shock wave to propagate into the low density medium and a rarefaction wave into the high density medium. These two flow regions are separated by a contact discontinuity.

At $x = 0$ and $x = 1$ reflecting boundary conditions are imposed. Analytic solutions can be obtained for the early phases of the evolution. This problem exhibits several interactions of nonlinear waves, shock reflection, shock merging, the interaction of a shock with a contact discontinuity, and the reflection of a rarefaction wave.

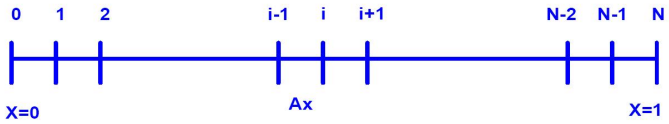

Figure 2. 1D discretization of Sod's 'Shock Tube' problem.

III.A.2. Step 1.1 Numerics

A finite differences scheme is used for the resolution of the 1D Euler equations. First, the computational domain is discretized in $N-1$ cells with a spatial increment Δx (see figure 2).

The numeric flux has to be evaluated at the interface $i+\frac{1}{2}$ between two consecutive nodes i and $i+1$, and the solution update is performed as

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x}(f^n_{\frac{i+1}{2}} - f^n_{\frac{i-1}{2}}) \tag{4}$$

where Δt is the time increment and the convective fluxes $f^n_{\frac{i+1}{2}}$ and $f^n_{\frac{i-1}{2}}$ in every time step n are evaluated using a Roe's scheme that reads as follows:

$$f^{Roe}_{\frac{i+1}{2}} = \frac{1}{2}(f_i + f_{i+1}) - \frac{1}{2}|A|(U_{i+1} - U_i) = \frac{1}{2}(f_i + f_{i+1}) - \frac{1}{2}\Sigma_{j=1}^{3}|\overline{\lambda_j}|\partial w_j r_j \tag{5}$$

where the first term of the equation (5) is evaluated using the value of the convective fluxes at the nodes i and $i+1$

$$f_i = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u H \end{pmatrix}_i, f_{i+1} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u H \end{pmatrix}_{i+1} \tag{6}$$

and the second term of the equation (5) is evaluated using the Roe's average variables. For this purpose, it is necessary a reconstruction of the conservative variables at the interface $i+1/2$ using the first derivative

$$U_L = U_i + \frac{\Delta x}{2}(\partial_x U)_i, \quad U_R = U_{i+1} - \frac{\Delta x}{2}(\partial_x U)_{i+1} \tag{7}$$

where L denotes the left state and R the right one. On the other hand, it is necessary to replace the value of the conservative variables for an average value. E.g. the conservative variables at the interface $i+1/2$ have the following values

$$\overline{R} = \sqrt{\frac{\rho_{i+1}}{\rho_i}}, \quad \overline{\rho} = \overline{R}\rho_i, \quad \overline{u} = \frac{\overline{R}u_{i+1} + u_i}{\overline{R}+1}, \quad \overline{H} = \frac{\overline{R}H_{i+1} + H_i}{\overline{R}+1}, \quad \overline{c} = \sqrt{(\gamma-1)(\overline{H} - \frac{\overline{u}^2}{2})} \tag{8}$$

where c states the sound velocity. The eigenvalues of the jacobian matrix A are

$$\lambda_1 = \overline{u}, \quad \lambda_2 = \overline{u} + \overline{c}, \quad \lambda_3 = \overline{u} - \overline{c}, \tag{9}$$

and the eigenvectors are

$$r_1 = \begin{pmatrix} 1 \\ \overline{u} \\ \frac{\overline{u}^2}{2} \end{pmatrix}, \quad r_2 = \frac{\overline{\rho}}{2\overline{c}} \cdot \begin{pmatrix} 1 \\ \overline{u} + \overline{c} \\ \overline{H} + \overline{uc} \end{pmatrix}, \quad r_3 = \frac{\overline{\rho}}{2\overline{c}} \cdot \begin{pmatrix} 1 \\ \overline{u} - \overline{c} \\ \overline{H} - \overline{uc} \end{pmatrix}, \tag{10}$$

Finally, the wave amplitudes are evaluated as

$$\partial w_1 = (\rho_{i+1} - \rho_i) - \frac{p_{i+1} - p_i}{\overline{c}^2}, \quad \partial w_2 = (u_{i+1} - u_i) + \frac{p_{i+1} - p_i}{\overline{\rho c}}, \quad \partial w_3 = (u_{i+1} - u_i) - \frac{p_{i+1} - p_i}{\overline{\rho c}}, \tag{11}$$

The selection of the algorithm for the convective flux computation as a critical subroutine for hardware implementation is justified by three key reasons:

- The performance studies made in CFD software, indicate that this subroutine consumes an important percentage (more than 50%) of the total time for the Euler equations resolution as will be detailed below.
- The independence of the algorithm with respect to the number of dimensions of the domain makes possible the utilization of the hardware accelerator for the resolution of 2D and 3D Euler equations.
- The industrial relevance that could be obtained applying this new technology to this classical problem.

III.B. Step 2: Design of a specific hardware accelerator

For the complete development of a reconfigurable hardware accelerator for CFD applications, a methodology is defined to guide the process:

1. **Performance analysis of the current CFD software implementation.** For the optimization and acceleration of algorithms it is necessary to make a preliminary analysis of the software to determine where the main bottlenecks are.
2. **Design and development of a CFD-specific HW-based system architecture** This step is aimed at the definition, design and development of the prototype of an FPGA-based accelerator card, directly connectable to current simulation platforms. For the practical implementation of the selected algorithm, the utilization of a commercial board is being considered. The following board characteristics are required to obtain maximal acceleration:
   - High number of FPGA modules to exploit the parallelism and to compute several nodes at the same time.
   - Ultra fast I/O communication interface to couple with the performance provided by the FPGA modules.
   - Low latency data access from the system memory.
3. **Development of specific hardware for the most time-consuming subroutines** The main objective of this step is to implement a library of cores (VHDL-coded), that is a set of independent modules with a fixed functionality, for the most common and time-consuming algorithms in the CFD codes. The detailed analysis of the CFD code aimed to find out which are the most-time consuming functions is performed in step 1 of the methodology. Although the main goal of this library is to provide support to accelerate the critical parts of the algorithms independently (i.e. the FPGA is configured depending on the actual application being performed), some effort is also required to provide common interfaces to at least some of the IPs so that they can be configured dynamically at runtime in the FPGAs (vendors claim that this can be done in a matter of milliseconds). The amount of success in this step depends mainly on the possibilities offered by the algorithms to provide a common interface with the rest of the FPGA circuitry (memory, I/O, etc.)
4. **Implementation of specific software for synchronization and communication** This step is intended to develop the software routines that support the synchronization and communication (including data exchange and protocols) between the code running in the PC and the hardware modules configured on the accelerator card.

   In the following sections, the previous steps of the methodology are presented in more detail and some considerations are made about the practical implementation.

III.B.1. Step 2.1: Performance analysis of the current implementation

The main goal of this phase is the analysis of the performance bottlenecks in order to obtain improvements in the overall performance through the acceleration of the most critical parts. This step is split into the following processes:

(a) Analysis of performance bottlenecks. This process is aimed at determining the most time and memory consuming algorithms in the set of applications considered.

In the proposed example, this step implies performing a specific analysis of the execution time for Roe computation function, over the total time required for the resolution of Sod's problem. The aim of this analysis is to confirm that the Roe flux computation function is a real bottleneck in the resolution of the Euler equations and therefore, its implementation in hardware will accelerate the overall process.

The result of the analysis is that the computation of the convective fluxes using a Roe scheme takes a 67% of the global time for the resolution of Euler equations in the one-dimensional Sod's problem. For more complex simulations, including 2D and 3D configurations, it is estimated that the computation of Roe's fluxes takes a percentage of 50%-60% over the total time for Euler resolution. This issue is due to the mayor complexity of the data structures for mesh representation in multidimensional domains.

From this analysis, it is deduced that the Roe flux computation takes an important percentage of the solver execution time, and the acceleration of this function using a hardware implementation could imply a great reduction of the time.

(b) Feasibility of the hardware implementation. A study about the feasibility of implementing in hardware the most critical blocks determined in the previous step is conducted. The following issues need to be addressed:

- Availability of efficient hardware implementations. In many cases, algorithms are implemented in software by using constructs and features that have no direct correspondence in hardware. Therefore, it is foreseen that a number of modifications in the algorithms will be required to adapt them to their hardware implementation. For example, formal functional transformations, typically to improve parallelism, compiler-based transformations, usually aimed at optimizing the handling of loops and other control structures, modifications aimed at the optimization of data structures and memory access patterns, substitution of complex operators by simplified versions more suitable for a hardware implementation and adaptation of high-level scheduling of tasks so that parallelism among them is improved and the overhead from hardware/software communications/synchronization is minimized.
  In the Roe flux computation algorithm, there are some divisions by constants. The division operation is not cheap (in terms of both performance and area requirements) when implemented in hardware (and neither in software). Therefore, the division operations are transformed into multiplications wherever is possible. Another important consideration is the identifications of common expressions in order to reuse the partial results and reduce the number of required resources. These issues are usually performed by a software compiler, but similar techniques can also be successfully applied for hardware generation.
- Degree of parallelism available in the different algorithms. This analysis provides a coarse estimation of the speedup that can be obtained from the hardware implementation of each particular block.
  In the proposed implementation, the degree of parallelism to be achieved is only limited by the amount of area available for hardware module synthesis and by communication bandwidth issues, because the algorithm itself has not a real limit on parallelism (only the grid size). Therefore, the Roe's flux computation could be performed for every node in the global grid (in the same iteration), if enough hardware modules could be replicated in the FPGA. However, in a general case, there could be limits in the degree of parallelism of an algorithm. In these situations, it could be necessary to apply techniques like loop-unrolling, often used in basic software compilers, to find the maximum available parallelism.
- Data structures and memory organization. It is necessary to characterize the access patterns to the memory structure implied by the different algorithms and data structures, as they may have a significant impact on the performance of the hardware accelerator.
  The numerical solution of the flow has data locality. This means that the computation to be performed at a certain location of the domain depends only on data that is located in a small neighborhood around it. Indeed, to compute the Roe's flux for node i, only the values

at nodes $i-1$ and $i+1$ are needed. This is the reason why communication costs are small with respect to the overall cost.
In addition, the extension of this design to a bi-dimensional or tri-dimensional system is easy because the flux computation algorithm involves always two nodes, and only small changes would have to be performed.

- Communication and synchronization requirements. The interface between the hardware and the software parts of the accelerator platform can easily become a bottleneck. Therefore, the hardware/software partitioning of the applications should also be guided by considerations regarding the hardware/software communication and synchronization requirements.
  The selection of the Roe computation module for hardware implementation is a key point in hardware-software partitioning. It is observed that it does not involve huge requirements of communication bandwidth, because only few data are required to perform each computation. In a first approach, the Roe's flux computation is performed in hardware and the other operations of the Sod's problem solver are performed in software. In this situation, an interface for hardware-software synchronization is developed. This interface includes software subroutines to control the evolution of the algorithm. For example, there is a software subroutine for detecting the end of an iteration, when the Roe flux computation of every node in the grid is finished and the solution update (in the software part) has to be executed.
- Precision requirements. Since the hardware implementation of the critical blocks is based on fixed-point arithmetic, some error is introduced with respect to the floating-point specification running in software[21,22] . Therefore, some bounds must be defined for such errors in order to guide the synthesis process.
  In the proposed practical implementation, the precision requirements are related to the numerical method selected for the resolution. Since Roe's method is a first order method, the error involved is always bound by a constant times the mesh size. The mesh size is computed as the value $1/N$, where N is the number of points in which the domain is decomposed. For example, for a domain of 10^3 points the error of the method should be smaller than a constant times $1/1000 = 10^{-3}$. This means that it is advisable to consider at least four or five fractional digits in the data representation. In the module developed, a precision of 10^{-5} has been considered for the tests performed, but the hardware module is designed using the precision as an input parameter, so it would be possible to consider other precision requirements without changes in the specification. However, an increment of the precision requirements would involve also an increment in the area and communication requirements, so some further changes might be necessary to achieve an optimal balance.
  The selection of an optimal wordlength for each operand involved in the computations is key to achieve huge speedups with FPGAs, and a detailed analysis has to be performed.
  In the following "Data representation specification" process, the work performed in the specification of an optimal data representation for this particular problem is presented in detail.

(c) Data representation specification. The selection of a proper data representation is a key task in the design of a hardware module with the objective to improve at maximum the performance without precision loss. Moreover, the data representation has a strong impact on the area needed for the implementation, and therefore on the feasibility of a practical hardware-software platform.
CFD tools typically handle floating point data in simple or double precision, which offers both large dynamic range and high precision for each numerical computation. However, for hardware implementations, the final form rarely uses a full-precision floating-point unit, given issues of silicon area, power, and speed. This creates the common and still awkward problem of transforming the application from its initial infinitive precision form, into a finite-precision fixed-point format.[21]
The two-fold problem for this complex task is how to choose the smallest word lengths for the fixed-point formats, and then how to validate that the format choices maintain the required numerical precision. A common solution is to rely on detailed simulations to find the dynamic range and precision of each operand. However, the input patterns for the simulation must be carefully selected. If a short or incorrectly distributed sequence of input patterns is selected, the results may fail to provide all the extreme values that real-life use will encounter. On the other hand, if a more rigorous detailed simulation strategy is used instead, the format optimization

process can become unduly expensive.

A more attractive solution is static analysis, which provides the extreme values and the maximum error of each operand, but does not require a potentially unbounded set of input patterns. The obvious approach for such a static analysis draws on techniques from interval arithmetic[23] , which replaces each scalar value with a bounded interval on the real number line. Interval arithmetic supports the simulation of the code by performing each operation of the algorithm in terms of intervals. Unfortunately, conventional interval arithmetic suffers from the problem of range explosion: interval results can be oversized as intervals may not capture the effect of crossed data dependencies in the algorithm. This is particularly harmful in iterative algorithms where the output ranges tend grow without bound.

A recent solution to this problem is a more sophisticated interval model called affine arithmetic[18,19,22] . The approach explicitly captures the correlations among operands, and dramatically reduces the level of oversize in the final intervals.

Another important aspect here is the optimization process to achieve the optimal wordlength for each operand in the algorithm. This process has to deal with a huge design space in both variables and computations, and the computational times required to get solutions could be considerable.

Currently, there are some tools for automatic floating point to fixed point transformation based on the above techniques (mainly simulation). One of them is the Matlab Floating-Point to Fixed-Point Transformation Toolbox that can automatically generate fixed-point MATLAB programs in terms of word-length (bit width) for a given floating-point program and error specification. This toolbox receives a pattern of input data and offers two optimization algorithms: gradient based algorithm and genetic algorithm[24] . Despite that the vendor claims that the resulting fixed-point specification is optimal, such claim should be taken with caution, since word-length optimization is an NP-complete problem and, therefore, no affirmation can be made about the optimality of the results obtained from an approach based on a very limited number of simulations. In general, it is enough to obtain a fixed-point specification that provides a good selection of word-lengths for the given error, and in the following it is assumed that the output of the Matlab Toolbox verifies this condition.

In the next sections, two approximations to the problem of data representation in the specific CFD area are presented. The first one is the possibility to implement a module that handles floating point data directly. This approach has become feasible in recent years with the rapid increase in the number of logic cells available in modern FPGA's. The second consideration for data representation is based on the conversion from floating-point formats to fixed-point formats. In this case, the input data has been analyzed to obtain an fixed point representation for the data using the current version of Matlab toolbox. The final purpose is to select the representation that provides the best balance between performance, precision and area for the specific problem.

Floating-point representation

First, the possibility to represent the data in a floating point format with simple precision (32 bits) is considered. This approach is based on the Xilinx Floating Point Operator v3.0 available from November of 2006 and incorporated into Xilinx ISE 8.2 software. The Floating Point Operator v3.0 is an IP core for handling floating point operations and it is configurable by the user from the CORE Generator tool. The operations available in the core are addition, subtraction, multiplication, division, square root and other operations for conversion between floating-point and fixed-point formats.

These operations are completely pipelined and are able to obtain a new result in every clock cycle. The latency of these operations depends on the selected data representation. For example, the latency for the multiplication is showed in table (1).

This Floating Point Operator is used in the design of the Roe flux computation module by implementing the expression in equation (12).

$$f_i = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u H \end{pmatrix}_i \quad f_{i+1} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u H \end{pmatrix}_{i+1} \tag{12}$$

Table 1. Floating point multiplication latency.

Fraction Width (bits)	Maximum latency
4-17	4
18-34	6
35-51	7
52-64	8

that corresponds to the first term of equation (5).

As previously mentioned, operations are completely pipelined and are able to obtain a new result in every clock cycle, although this can be configured by the user through an input parameter. This parameter describes the minimum number of cycles that must elapse between inputs. A value of 1 allows operands to be applied on every clock cycle, and results in a fully-parallel circuit. A value greater than 1 enables hardware reuse. The number of slices consumed by the core reduces as the number of cycles per operation is increased. A value of 2 approximately halves the number of slices used. A fully sequential implementation is obtained when the value is equal to fraction width+1 for the square-root operation, and fraction width+2 for the divide operation[15] .

In this particular case, and since floating-point module supports that all operations are pipelined and return a new result every cycle, it is decided to implement a module for Roe's computation able to return data every cycle too. This decision has an impact on the total area, because some operators have to be duplicated in order to support a new operation every cycle, but a maximum throughput will be obtained from the module.

The approximation selected to implement expression (5) can be observed in figure (3). The

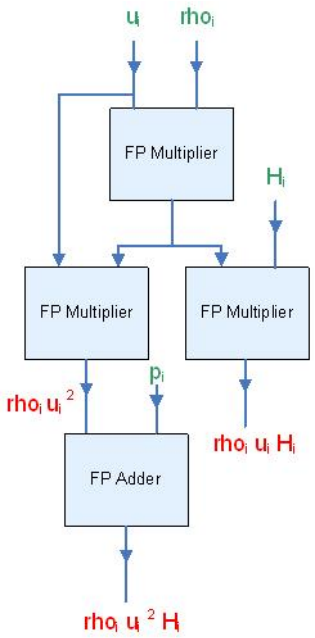

Figure 3. Hardware implementation of expression (12).

behavior of this circuit was verified with the ModelSim simulator and synthesized into a Xilinx Virtex II Pro (xc2vp30-6ff896). The final synthesis report (obtained with Xilinx ISE 8.2 synthesis tool) is showed in table (2).

As can be observed from the synthesis report, a single implementation of the expression (12), that involves 3 multiplications and 1 addition, uses 15% of the multipliers available in the selected FPGA.

Since the complete computation of the Roe fluxes would need around 24 multiplications, 27

Table 2. Synthesis report from Xilinx ISE.

Logic utilization	Utilization percentage
Number of slices	10%
Number of 18x18 multipliers	15%

additions, 6 divisions, and 2 square roots, it is not possible to implement the whole module using this architecture due to area limitations.

However, there are still several alternatives for the implementation of Roe computation into hardware:

i. Selection of a modern FPGA chip like Xilinx Virtex5, which has more logic cells available. This alternative might permit the implementation of a Roe computation module in simple precision format but not in double precision, and moreover, it would not be possible to synthesize several modules in the chip, and therefore no parallelization would be achieved.

ii. Change the fully-parallel selected architecture into a fully-sequential or a balanced architecture. This decision would reduce the number of slices consumed at the cost of increasing the number of cycles between two consecutive solutions, meaning that the circuit would not be able to return a new result every cycle.

iii. Implementation using a fixed-point format representation. Obviously, the transformation of the floating-point algorithm into a fixed-point format requires a substantial effort, and the key problems are how to choose the smallest bit-level number formats, and then how to validate that the format choices maintain the necessary level of numerical precision. However, the use of this representation implies minimum area requirements (and, therefore, maximum parallelization) while maintaining the same precision.

In addition, the fixed-point implementation will permit the synthesis of several modules and the computation of several nodes at the same time, thus increasing the overall performance.

In this paper, a fixed-point representation is finally selected for the Roe computation module. However, floating-point arithmetic should not be discarded as an alternative for implementations using FPGA technology in the near future.

Fixed-point representation

As previously mentioned, the most important phase in the implementation of the Roe flux computation in a fixed-point format is the transformation from a floating-point to a fixed-point data representation.

This phase is one of the most time-consuming phases in the hardware design process. The floating-point specifications of the previous step must be converted into fixed-point specifications according to the error requirements defined during code analysis. The goal is to obtain an optimized data representation for each operand that guarantees the required precision with the minimum area cost and the maximum performance. Since the design space (i.e. possible combinations of word-lengths in the data representation of all variables) is so huge, standard simulation-based approaches usually take very long computation times despite only covering a very small fraction of the design space, especially in cases when the specification cannot be partitioned for analysis due to required data dependencies.

This transformation is completely dependent on the algorithm to be analyzed. Therefore, a study of the range of data in the solution of the Euler solver has been conducted and its results are explained below.

For this purpose, a test case for the Sod's shock tube problem in subsonic case has been executed to obtain the values of density, velocity, pressure, and enthalpy in every node of the computational grid. These values are the input for the hardware accelerator module.

The range of the values of density obtained from the test case is displayed in figure (4). The left figure represents the solution of the selected problem for the density value, while the right figure displays the histogram of density values in order to show its range of values.

The density values are distributed between 0 and 1, as expected. This seems the program operates with values normalized with respect to the value of density in the infinity (constant).

This resulting dynamic range has an important impact on the selection of an optimal representation for the data, because there are not big distances between the values to be considered, and there are not big numbers or very small ones either. In addition, due to the normalized values, the length of the mantissa is more important to maintain the required precision than the exponent length. Therefore, the fixed-point representation seems to adapt very well to this algorithm.

Once the range of input values and the algorithm are known, the transformation from a floating point to a fixed point algorithm can proceed. As explained in the data representation section, there are several alternatives for the transformation: simulation, interval arithmetic, affine arithmetic, etc. In particular, the simulation-based Floating-Point to Fixed-Point Transformation Matlab Toolbox[25] is used in this work. However, it should be mentioned that the authors are developing in-house analytic-based tools that should be operational in the near future, in order to substantially reduce computation times.

This toolbox requires a pattern of data that are the input to the algorithm. The inputs to the proposed algorithm are U_i (the vector of variables at node i, which is composed by ρ_i, the density at node i, u_i, the velocity at node i, and H_i, the enthalpy at node i), and U_{i+1} (the vector of variables at node $i+1$, which is composed by ρ_{i+1}, the density at node $i+1$, u_{i+1}, the velocity at node $i+1$, and H_{i+1}, the enthalpy at node $i+1$). The ranges of these data are obtained from the test case and their values are approximated by means of probability density functions (like normal, uniform, gaussian, chi-square...) between the maximum and minimum values. The sequences of input data for the algorithm are obtained as samples following these probability density functions.

Once the generation of input data is properly defined, the toolbox optimization procedure is started for a desired error of 10^{-5}. This computation takes a long time to return the data representation for each operand in the algorithm, as it is based on repeated simulations of the algorithm for different word-length configurations. The solution is the minimum number of bits needed for the representation of each operand that maintain the required precision.

III.B.2. Step 2.2: Design and development of a CFD-specific HW-based system architecture

This phase includes the definition, design and development of the prototype of an FPGA-based accelerator card, directly connectable to current simulation platforms.

In the practical case discussed here, a commercial development board has been selected for the planned implementation because the algorithm does not have special requirements to achieve a significant improvement of performance. The board selected is a Xilinx Virtex II Pro development kit from AVNET. The main features of this board are:

- Virtex-2 Pro XC2VP30 FPGA with 30,816 Logic Cells, 136 18-bit multipliers, 2,448Kb of block RAM, and two PowerPC Processors
- DDR SDRAM DIMM that can accept up to 2Gbytes of RAM
- 10/100 Ethernet port
- USB2 port
- PCI port
- Compact Flash card slot

However, for the implementation of more complex problems, and for obtaining an optimal performance, it can be necessary to design a specific architecture for the acceleration board (with specific memory and communications requirements). In these cases, it would be necessary to define the number and type of FPGAs in the board, the interfaces that should be supported in order to allow high speed data transfers, the system control and monitoring, the procedures for real-time FPGA programming and functional testing of the modules developed, and the organization and size of on-board memory in order to support high access rates and also fast FPGA configuration and programming (from ROM).

III.B.3. Step 2.3: Development of specific hardware for the most time-consuming subroutines

The goal of this phase is to provide a library of cores (VHDL-coded) for the most common and time-consuming algorithms in the CFD codes. To achieve such objective, this step is split into the following activities:

(a) Functional specification of the IP cores. The goal is to obtain specifications that can be validated in terms of their functionality and can be used as a reference during the hardware design process. In addition to the computational blocks, other blocks that are required as part of the hardware implementation of the cores, like synchronization and communication routines and primitives, must also be specified in this phase.

In our case-study, the starting point is the software implementation of the Roe flux computation algorithm. However, as a result of the previous steps, some changes are performed in the original algorithm in order to be implemented in hardware, as for example, transformations of complex operations. Therefore, the original software implementation is modified to validate the functionality of the hardware module and to compare the software results with the data obtained from the hardware-software approach.

(b) Architecture definition and synthesis. The goal in this phase is to obtain a synthesizable RTL description of the blocks that can be fed into commercial synthesis tools to obtain the netlists corresponding to all the IP cores. The first step is the definition of the architecture of each IP. Some of the considerations that must be taken into account during this phase are:

- Good balance between area and latency.
- Feasible communications rate.
- Optimization strategies based on the features of the specific problem, like local data distribution in grids, the order of the computation of fluxes, etc.
- Use of embedded resources (computational -DSP- and storage) provided by the FPGAs.
- Whenever possible, common interfaces between IP cores to support dynamic configuration at run time.

In the practical approximation, the hardware flux computation module has the following input-output parameters, as displayed in figure (5).

- The input U_i is the vector of variables at node i. It is composed by ρ_i, the density at node i, u_i, the velocity at node i, and H_i, the enthalpy at node i.
- The input U_{i+1} is the vector of variables at node $i+1$. It is composed by ρ_{i+1}, the density at node $i+1$, u_{i+1}, the velocity at node $i+1$, and H_{i+1}, the enthalpy at node $i+1$.
- The output of the module is a vector that represents the value of the flux at point $i+1/2$ as expressed in equation (13).

As an additional optimization, the variables related to one of the points involved, could be stored in the internal FPGA memory for the next computation, and therefore, a reduction of the communication requirements would be obtained, because only three variable per flux computation should be transmitted.

$$f^{Roe}_{\frac{i+1}{2}} = \frac{1}{2}(f_i + f_{i+1}) - \frac{1}{2}\Sigma_{j=1}^{3}|\overline{\lambda_j}|\partial w_j r_j \tag{13}$$

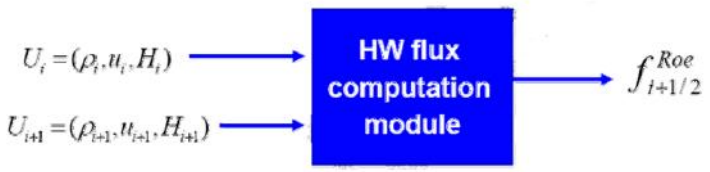

Figure 5. I/O module specifications.

In summary, the architecture is selected to obtain a good balance between area and latency, taking into account the communication bandwidth limit of the development board.

Once the representation of the operands is known and the module architecture has been selected, the implementation of the hardware module can start. The automatic programming of hardware circuits is usually performed using Hardware Description Languages like VHDL or Verilog. These languages permit the description of the hardware at RTL level of abstraction. This means that they require some skills to specify a desired high-level behaviour because the designer has to be aware of clock timings and other low-level restrictions, but, on the other hand, the designer can control and optimize each part of the circuit. In particular, the specification of the hardware part in this work is written using the VHDL language.

It should also be mentioned that other languages for hardware description at the behavioral level have appeared recently. These languages called high level synthesis languages permit the generation of circuits from a behavioral specification. However, currently these tools are still in their infancy because they do not provide the optimal solution in terms of area requirements, so they are basically used for functional validation.

(c) Placement and routing of IP cores. Once the IP cores are synthesized using commercial tools, the final design tasks are the placement and routing of the modules. These tasks are critical to achieve the expected performance and, although commercial tools can provide fully automated results, it is usually the case that manual intervention by engineers is required to avoid substantial degradations in both, performance and area usage.

III.B.4. Step 2.4: Implementation of specific software for synchronization and communication

Finally, this step includes the development of software routines to support the synchronization and communication between the PC and the accelerator card.

The phase is split into the following processes:

(a) Definition of mechanisms for hardware/software interaction. This first step deals with the identification of the different contexts during the execution of CFD applications when interactions between software and hardware might take place, and the definition of the mechanisms to support them.

(b) Implementation of software primitives and routines. This stage deals with the implementation of the mechanisms defined in the previous step.

(c) Validation of the hardware/software interaction mechanisms.

(d) Integration into a mixed hardware/software platform. Integration of the accelerator card prototype into a simulation platform (single PC and cluster) and evaluation of the overall performance.

In the application analyzed in this work, the software subroutines for the data communication between PC-host and acceleration board are implemented.

III.C. Step 3: Practical experiments

In this section the results of the synthesis of the hardware module and its expected performance are presented. First, the technology selected for the synthesis and its main features are briefly described, since the final implementation depends on the selected FPGA. Second, the synthesis results related to area used and

frequency obtained are shown. Finally, an analysis of the expected performance in a mixed hardware-software simulation platform is presented.

III.C.1. Step 3.1: Experiment description

Since the experiments are performed in both Virtex II and Virtex IV FPGAs, the main capabilities of these chips are briefly described.

The Xilinx Virtex II Pro (xc2vp30) contains 30,000 logic cells, 2MB BRAM, 136 18x18 multipliers, and 2 PowerPC processors.

The Xilinx Virtex 4 (xc4vlx200) contains 200,000 logic cells, 6MB BRAM, embedded PowerPC processors, and Multi-Gigabit Serial I/O.

The synthesis report for the Roe flux computation hardware module on a Virtex II (xc2vp30) is displayed in table (3).

Table 3. Synthesis report from Xilinx ISE.

Number of slices	21%
Number of 18x18 multipliers	17%
Minimum period	8.156ns
Maximum frequency	122.602MHz
Latency	4 cycles

These synthesis results indicate that it is only possible to implement a maximum of four parallel Roe flux computation modules in this type of FPGA. However, if a Virtex IV technology is employed, the level of parallelism is increased. Indeed, the synthesis report for 10 hardware modules for Roe flux computation on a Virtex IV (xc4vlx200) is displayed in table (4), showing that around 20 modules could be implemented in a single FPGA. For the synthesis on the Virtex IV FPGA, embedded DSP modules (similar to embedded 18x18 multipliers in the Virtex II device) are not used in this test, but the utilization of these modules would reduce the amount of slices needed.

Table 4. Synthesis report from Xilinx ISE.

Number of slices	47%
Minimum period	11.313ns
Maximum frequency	88.394MHz

III.C.2. Step 3.2: Numeric results: CPU vs. FPGA

In order to obtain numerical performance results, the Sod's problem is considered for a mesh composed of 1.000.000 nodes. The computation times of Roe convective fluxes computation for all the nodes are:

$$\begin{aligned} &\text{Time for Euler resolution (1 iteration)} = 0.62s. \\ &\text{Time consumed in Roe's flux computation (1 iteration)} = 0.42s. \end{aligned} \tag{14}$$

As previously mentioned, the time consumed by the Roe's flux computation is around the 67% of the total time for the resolution of the quasi 1D Euler equations.

If the analysis for the hardware implementation on the Virtex II Pro FPGA is performed, the execution time is:

$$\begin{aligned} &\text{HW Module execution time} = 32.62ns \\ &\text{(latency} = 4 \text{ cycles)} \end{aligned} \tag{15}$$

The equation (15) expresses that the time for a computation of the Roe convective flux for one computational node of the grid is 32.62 ns. So, the total time required by a single hardware module is obtained

multiplying this value by the number of nodes in the grid (16).

$$32.62ns * 1.000.000nodes = 0.32s. \tag{16}$$

Therefore, the time for Roe's flux hardware computation of every point in the mesh is 0.32 s. This means that using a single hardware module would produce an improvement in the overall algorithm with respect to the software implementation.

In addition, these algorithms usually need to execute several iterations to get converged results. If 5000 iterations are considered, the performance comparison between software and hardware implementation can be observed in expressions (17) and (18).

$$SW : 0.42 * 5000 = 2100s. => 35min. \tag{17}$$

$$HW : 0.32 * 5000 = 1600s. => 26min.40s. \tag{18}$$

The selected architecture tries to find a balance between performance and area requirements in the available technology, and therefore, a non-fully pipelined architecture has been selected. However, if the latest Virtex 5 (with more logic cells) was the target technology, a fully pipelined architecture could be designed and the module would be able to return a new data every clock cycle. Estimations about the additional gains and the expected performance with a fully pipelined architecture are displayed in (19).

$$HW : 0.00815 * 5000 = 40.75s. \tag{19}$$

In addition to the previous results that show a performance improvement, the huge gains are obtained when exploiting the parallelism inherent in the architecture. In next section, the results obtained for the synthesis of several modules in a Virtex IV FPGA are shown.

III.C.3. Step 3.3: Analysis of the expected performance

Table (5) shows the comparison between CPU and FPGA execution times when 20 hardware modules are synthesized in a Virtex IV FPGA (times obtained for 5000 iterations).

Table 5. Synthesis of 20 hardware modules in a Virtex IV FPGA.

Platform	Execution time
SW-CPU:	35 min.
HW-FPGA	8.15 s.

Finally, the expected performance when implementing several modules into a "simulation board" like (6) composed by 32 Virtex 4 FPGAs and 20 modules in each FPGA is presented in table (6)

Table 6. Synthesis of 20 hardware modules in a simulation board with 32 Virtex IV.

Platform	Execution time
SW-CPU:	35 min.
HW-FPGA	0.25 s.

It is necessary to remark that bandwidth requirements are going to be a performance bottleneck with the currently available technologies. Therefore, it is necessary to achieve a good balance between bandwidth requirements and performance. In addition, alternative communication technologies such PCI-Express or HyperTransport will be considered in a near future.

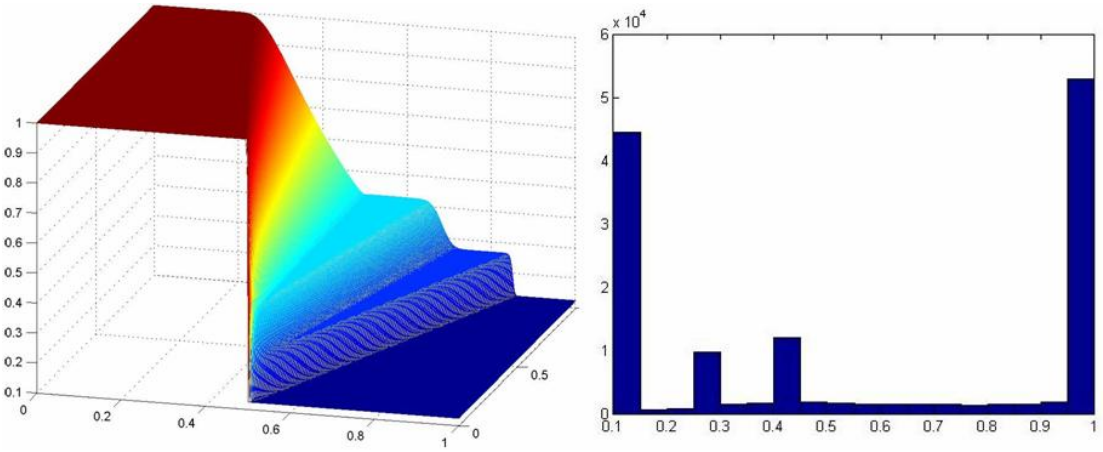

Figure 4. Results obtained for the rho variable (density)

IV. Conclusions

The aim of this paper is to introduce the concept of hardware acceleration in the CFD applications area. The feasibility of the hardware implementation has been demonstrated, and a methodology has been proposed as a guide for the implementation of hardware accelerators.

A practical implementation for the quasi 1D Euler equations has been presented with numerical results of the expected performance, showing that speedups above two orders of magnitude can be expected from the used of FPGA-based hardware modules.

Future efforts would be performed to achieve a complete implementation with optimal balance between area and performance and a simulation board with specific architecture (memory, communication technology...) will be designed to test the acceleration into a simulation platform.

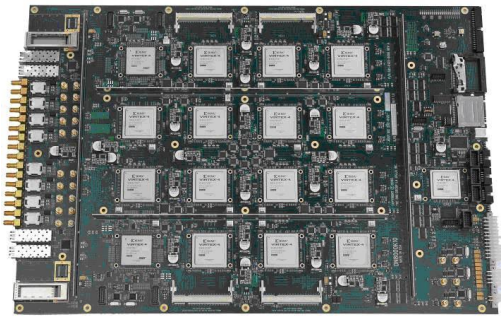

Figure 6. Multicore FPGA board.

In addition, other algorithms in aeronautical analysis and design will be analyzed for a suitable hardware implementation of their most time-consuming subroutines and functions.

V. Acknowledgments

The research described in this paper made by INTA's researchers has been supported under the INTA activity termofluidodinmica (INTA's code: IGB4400903) and by the Spanish Ministry of Education and Science (MEC) under Projects DOMINO (CIT-370200-2005-10) and AMEBA3 (TEC-2006-13067-C03-03). Also, this work has been partially supported by the grant BFM2002-03345 of the Spanish MEC.

References

[1]Baley, F. R. and Simon, H. D., "Future directions in computing and CFD," *AIAA American Institute of Aeronautics and Astronautic*, 1992.

[2]Bolz, J., Farmer, I., Grinspun, E., and Schreoder, P., "Sparse matrix solvers on the GPU: conjugate gradients and multigrid," *Transactions on Graphics (TOG)*, 2003.

[3]Krger, J. and Westermann, R., "Linear algebra operators for GPU implementation of numerical algorithms," *Transactions on Graphics (TOG)*, 2003.

[4]Moreland, K. and Angel, E., "The FFT on a GPU," *in Proceedings of HWWS*, 2003.

[5]Todman, T., Constantinides, G., Wilson, S., Mencer, O., Luk, W., and Cheung, P., "Reconfigurable Computing: Architectures and Design Methods," *IEEE Proc. Comput. Digit. Tech. Vol. 152, No. 2*, 2005.

[6]Luk, W. and Cheung, P., *Configurable Computing*, 'Electrical Engineer's Handbook', Academic Press, 2004.

[7]Compton, K. and Hauck, S., "Reconfigurable Computing: a Survey of Systems and Software," *ACM Computing Surveys*, 2002.

[8]G. Caffarena, C. Pedreira, C. C. S. B. O. N.-T., "FPGA Acceleration for DNA Sequence Alignment," *Journal of Circuits, Systems and Computers, Vol. 16, no. 2*, 2007.

[9]Frigo, J., Palmer, D., Gokhale, M., and Popkin-Paine, M., "Detection using Reconfigurable Computing Hardware," *Proceedings of the Symposium on Field-Progammable Custom Computing Machines. IEEE Computer Society Press*, 2003.

[10]Styles, H. and Luk, W., "Customising Graphics Applications: Techniques and Programming Interface," *Proceedings of the Symposium on Field-Programmable Custom Computing Machines, IEEE Computer Society Press*, 2000.

[11]Puttegowda, K., Worek, W., Pappas, N., Danapani, A., and Athanas, P., "A Run-time Reconfigurable System for Gene-sequence Searching," *Proceedings of the International VLSI Design Conference*, 2003.

[12]Mahmoud, B., Bedoui, M., Raychev, R., and Essabbah, H., "Nuclear medical image treatment system based on FPGA in real time," *International Journal of Signal Processing*, 2004.

[13]Castro-Pareja, C. R., Dandekar, O. S., and Shekhar, R., "FPGA-based real-time anisotropic diffusion filtering of 3D ultrasound images," 2005.

[14]Makino, J. and Koko, E., "Performance evaluation and tuning of GRAPE-6-towards 40 "real" Tflops," *In Proceedings of the ACM/IEEE SC2003 Conference, Phoenix, Arizona*, 2003.

[15]"Floating Point Operator v3.0," Tech. rep., Xilinx, 2006.

[16]Xilinx, "Virtex-4 Family Overview DS112(v1.5)," 2006.

[17]Strenski, D., "Computational Bottlenecks and Hardware Decisions for FPGAs," *FPGA and Structured ASIC Journal*, 2006.

[18]Fang, C., Puschel, M., and Chen, T., "Toward Efficient Static Analysis of Finite Precision Effects in DSP Applications via Affine Arithmetic Modeling," *Design Automation Conference (DAC)*, 2003.

[19]Fang, C., Puschel, M., and Chen, T., "Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs," *International Conference on Computer Aided Design (ICCAD)*, 2003.

[20]Menard, D. and O.S, "Automatic Evaluation of the Accuracy of Fixed-Point Algorithms," *Design, Automation and Test in Europe (DATE))*, 2002.

[21]Sung, W. and Kum, K.-I., "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE Trans. Signal Processing, vol. 43, pp. 3087.3090*, 1995.

[22]Carreras, C.; Lopez, J. N.-T. O., "Bit-width selection for data-path implementations," *Proceedings on System Synthesis 12th International Symposium Page(s):114 - 119*, 1999.

[23]Moore, R. E., *Interval Analysis*, Prentice-Hall, 1966.

[24]Kyungtae Han, B., "Floating-Point to Fixed-Point Transformation Toolbox," Tech. rep., DLR, 2006.

[25]Han, K. and Evans, B. L., "Floating-Point to Fixed-Point Transformation Toolbox," http://users.ece.utexas.edu/~bevans/projects/wordlength/converter/index.html#GPL.